Unions of Disjoint NP-Complete Sets

Christian Glaßer[*] John M. Hitchcock[†] A. Pavan[‡] Stephen Travers[*]

Abstract

We study the following question: if A and B are disjoint NP-complete sets, then is $A \cup B$ NP-complete? We provide necessary and sufficient conditions under which the union of disjoint NP-complete sets remain complete.

1 Introduction

A disjoint NP pair is a pair of languages (A, B) such that both A and B are in NP and are disjoint. Given a disjoint pair (A, B), we can view strings from A as "yes" instances and strings from B as "no" instances. We are interested in an algorithm that accepts all instances from A and rejects all instances from B. Viewed this way, disjoint pairs are an equivalent formulation of promise problems, where $A \cup B$ is the promise. Promise problems were introduced by Even, Selman, and Yacobi [ESY84].

Promise problems and disjoint pairs arise naturally in many scenarios such as the study of complete problems for semantic classes and the study of hardness of approximation problems. In some instances promise problems more precisely capture the underlying computational problem rather than decision problems. Sometimes unresolved questions about complexity classes can be answered by considering promise versions of complexity classes. For example, we know that Promise-MA does not have fixed polynomial-size circuits whereas we do not have an analogous result for the class MA [San09]. For a recent survey on promise problems we refer the reader to [Gol06].

In addition to be able to capture several natural computational problems, disjoint pairs arise naturally in the study of public key cryptosystems and propositional proof systems. The computational problem capturing a public key cryptosystem can be formulated as a disjoint NP pair (A, B) [ESY84, GS88]. A separator of such a pair (A, B) is a set S with $A \subseteq S$ and $S \subseteq \overline{B}$. The class of pairs (A, B) whose separators do not belong to P are called P-inseparable pairs. The existence of P-inseparable disjoint NP pairs is closely related the existence of secure public key cryptosystems [ESY84, GS88]. Grollmann and Selman [GS88] showed that if $\mathrm{P} \neq \mathrm{UP}$, then there exist P-inseparable disjoint NP pairs. More recently Fortnow, Lutz, and Mayordomo [FLM10] showed that if NP does not have p-measure zero, then P-inseparable disjoint NP pairs exist.

Works of Razborov [Raz94] and Pudlak [Pud01] show that disjoint NP pairs are also closely related to the study of propositional proof systems. Razborov identified a canonical disjoint NP pair

[*]Julius-Maximilians-Universität Würzburg, {glasser,travers}@informatik.uni-wuerzburg.de

[†]Department of Computer Science, University Wyoming. Research done while this author was on sabbatical at CWI and was supported in part by an NWO travel grant and by NSF grants 0652601 and 0917417. jhitchco@cs.uwyo.edu

[‡]Department of Computer Science. Iowa State University. Research supported in part by NSF grants 0830479 and 0916797. pavan@cs.iastate.edu

$(\mathrm{SAT}^*, \mathrm{REF}_f)$ for every propositional proof system f. Here SAT^* is a padded version of SAT and REF_f is the set of all formulas that have short proofs of unsatisfiability with respect to f. Glaßer, Selman, and Zhang [GSZ07] showed that for every disjoint NP pair (A, B) there is a propositional proof system f such that its canonical pair $(\mathrm{SAT}^*, \mathrm{REF}_f)$ is many-one equivalent to (A, B). Thus disjoint NP pairs and propositional systems have identical degree structure.

There is a close relation between disjoint NP pairs and pairs whose both components are NP-complete. For example, there is a P-inseparable disjoint NP pair if and only there is a P-inseparable pair whose both components are NP-complete [GS88]. We also know that if there is a complete pair for DisjNP, then there is such a pair where both components are NP-complete [GSS05].

In this article we focus on disjoint pairs whose both components are NP-complete. We investigate the following question: let (A, B) be a disjoint NP pair such that both A and B are NP-complete. Is the union $A \cup B$ NP-complete? This question was first explicitly raised by Selman [Sel88].

Apart from its connections to the study of public key cryptosystems and propositional proof systems, our question is also of independent interest. We are interested in a simple closure property of NP-complete sets—closure under disjoint unions. It is known that every NP-complete set can be split into two disjoint NP-complete sets [GPSZ08]. Here we are raising the converse question, is the combination of every two disjoint NP-complete sets NP-complete?

Glaßer et al. [GSTW08] showed that if A and B are disjoint NP-complete sets, then $A \cup B$ is complete via strong nondeterministic Turing reductions. They also showed that if NP differs from co-NP at almost all lengths, then $A \cup B$ is many-one complete via P/poly-reductions. If we consider disjoint Turing complete sets, we know a little more. Glaßer et al. [GPSS06] showed that if $\mathrm{UP} \cap \text{co-UP}$ contains bi-immune sets, then there exist disjoint Turing complete sets whose union is not Turing complete.

The above mentioned results do not seem to shed light on the question of whether unions of disjoint NP-complete sets remain NP-complete (under polynomial-time many-one reductions). To date, we do not know of any reasonable hypothesis that either provides a positive answer or a negative answer to this question. In this paper we provide necessary and sufficient conditions under which the union of disjoint NP-complete sets remain NP-complete. We consider two statements and show that one of the statements yields a positive answer to our question, whereas the other statement yields a negative answer.

Our statements relate to the complexity of $\overline{\mathrm{SAT}}$. Let us assume that NP differs from co-NP, thus there is no NP-algorithm for $\overline{\mathrm{SAT}}$. Could it still be the case that there is an NP-algorithm that solves $\overline{\mathrm{SAT}}$ in some "average-case/approximate" sense? Let B be a set in NP that is disjoint from SAT. We can view B as an "approximate/average-case" NP-algorithm for $\overline{\mathrm{SAT}}$. Since B does not coincide with $\overline{\mathrm{SAT}}$, there must exist unsatisfiable formulas on which B errs. How easy/hard is it to produce such instances? Any machine that produces instances on which B differs from $\overline{\mathrm{SAT}}$ is called a *refuter*.

Given B, what is the complexity of the refuter? We can make two easy observations. If B can be decided in time 2^{n^k}, then there exists a refuter that runs in time $O(2^{n^k}2^n)$. Using the fact that B is in NP we can also design a $\mathrm{P}^{\Sigma_2^{\mathrm{P}}}$ refuter. Can the complexity of these refuters be reduced further? We show that if the complexity of such refuters can be improved to polynomial-time, then unions of disjoint NP-complete sets remain NP-complete. On the other hand, we show that if the complexity of the refuters can not be reduced, then there exist disjoint NP-complete sets whose union is not NP-complete. More precisely, we show that if there exists a $B \in \mathrm{NP}$ that is disjoint from $\overline{\text{SAT}}$ such that any refuter for B must take 2^{2n} time, then there exist disjoint NP-complete sets whose union is not NP-complete.

The notion of refuters can be made precise by using *distinguishers* and *pseudo-classes*. These notions were first formally defined by Kabanets [Kab01]. These concepts have been proved to be useful in learning theory [JS05], in the study of heuristics for solving NP-complete problems, and in derandomization [IW01, Kab01]. In this paper, we provide yet another instance where such concepts seem to be useful.

2 Preliminaries

Given two languages A and B, $A\Delta B$ denotes the symmetric difference between A and B. A refuter R is a deterministic Turing machine that on an input of length n outputs a string of length at least n.

Definition 1. [Kab01] Let L and L' be two languages and R be a refuter. We say that R distinguishes L from L' if for infinitely many n, $R(1^n)$ outputs a string (of length $\geq n$) from $L\Delta L'$. A refuter R almost everywhere distinguishes L from L' , if for all but finitely many n, $R(1^n)$ outputs a string (of length $\geq n$) from $L\Delta L'$.

If a refuter R does not distinguish L from L', then for all but finitely many n, $R(1^n) \in (L \cap L') \cup (\overline{L} \cap \overline{L'})$. If a refuter does not almost everywhere distinguish L from L', then for infinitely many n, $R(1^n) \in (L \cap L') \cup (\overline{L} \cap \overline{L'})$.

Now we mention our statements.

Statement 1. There is a language $L \in$ NP that is disjoint from SAT and no 2^{2n}-time bounded refuter can distinguish $\overline{\text{SAT}}$ from L.

Informally, this means that no 2^{2n}-time bounded machine can output strings on which L differs from $\overline{\text{SAT}}$.

Statement 2. For every language $L \in$ NP that is disjoint from SAT, there is a polynomial-time refuter that almost everywhere distinguishes L from $\overline{\text{SAT}}$.

This statement implies that for every language $L \in$ NP that is disjoint from SAT, there is a polynomial-bounded refuter R such that $R(1^n)$ outputs a string of length $\geq n$ at which L differs from $\overline{\text{SAT}}$.

Observe that if we replace P with 2^{2n} and remove the phrase "almost everywhere" from Statement 2, then it would be a converse to Statement 1.

Main Theorem 1: If Statement 1 is true, then there exist disjoint NP-complete sets whose union is not NP-complete.

Main Theorem 2: If Statement 2 is true, then unions of disjoint NP-complete sets are NP-complete.

3 Main Theorems

We will show that if Statement 1 is true, then there exist two disjoint NP-complete sets whose union is not NP-complete. On the other hand, we show that if the Statement 2 is true, then unions of disjoint NP-complete sets remain NP-complete.

Let A be an NP-complete set and B be a set in NP that is disjoint from A. Let $A' = 0A \cup 1B$, and $B' = 1A \cup 0B$. Both A' and B' are NP-complete and are disjoint. The set $A' \cup B'$ is NP-complete if and only if $A \cup B$ is NP-complete. Thus we have the following observation.

Observation 3.1. *There exist two disjoint* NP*-complete sets whose union is not* NP*-complete if and only if there exists an* NP*-complete set A and a set B in* NP *that is disjoint from A such that $A \cup B$ is not* NP*-complete.*

Theorem 3.2. *If Statement 1 is true, then there exist two disjoint* NP*-complete sets whose union is not* NP*-complete.*

Proof. Let L be a language in NP that is disjoint from SAT and for every 2^{2n}-time bounded refuter R, for all but finitely many n, $R(1^n) \in \text{SAT} \cup L$ (note that $\text{SAT} \cap L = \emptyset$ and $R(1^n) \notin \overline{\text{SAT}} \Delta L$ implies $R(1^n) \in \text{SAT} \cup L$). We exhibit an NP-complete set A and a disjoint set B in NP such that $A \cup B$ is not NP-complete.

Since $L \in \text{NP}$, there is a constant $k \geq 1$ such that L can be decided in time 2^{n^k}. Let $t_1 = 2$, and $t_{i+1} = t_i^{k^2}$.

Before we present a formal proof, we provide the main ideas behind the proof. Let us partition Σ^* into blocks $B_1, B_2, \cdots$ such that

$$B_i = \{x \mid t_i^{1/k} \leq |x| < t_i^k\}.$$

Note that B_i is disjoint from B_{i+1} as the length of every string from B_i is less than t_i^k and every string from B_{i+1} has length at least $t_{i+1}^{1/k} = (t_i^{k^2})^{1/k} = t_i^k$. Let us take $L_1 = L \cap (\cup_i B_{2i})$ and L_2 to be $L \cap (\cup_i B_{2i+1})$.

To better express the intuition, we make the following assumption: There exist infinitely many strings of length t_{2i} (for some $i > 0$) and do not belong to L. By the definition of L_1, these strings do not belong to L_1.

Suppose that there is a many-one reduction f from L_1 to L_2. We will first argue that by using this reduction, there is a procedure that outputs infinitely strings (of length n) that are not in L in time less than 2^{2n}. Let us fix i. Consider a string x of length t_{2i} that is not in L. Recall that x lies in block B_{2i}. What does $f(x)$ look like? There are three possibilities: $f(x)$ remains in block B_{2i}, $f(x)$ is in block B_j for some $j < 2i$, or $f(x)$ lies in block B_j for some $j > 2i$.

Suppose $f(x)$ lies in block B_{2i}. Observe that $L_2 \cap B_{2i}$ is empty. This immediately implies that $f(x)$ does not belong to L_2 and thus x does not belong to L_1 (and thus x is not in L). If any string of length t_{2i} that is not in L is mapped into block B_{2i}, then one can find and output such a string in time $2^{2t_{2i}}$.

Now suppose $f(x)$ lies in block B_j and $j < 2i$. Since every string from block B_j is of length at most $t_{2i}^{1/k}$, we can decide whether $f(x)$ belongs to L_2 or not in time less than $2^{t_{2i}}$. Thus if any string of length t_{2i} that is not in L is mapped into block B_j ($j \leq 2i$), then one can find and output such a string in time $2^{2t_{2i}}$.

Now suppose that for every string x of length t_{2i} that is not in L, $f(x)$ lies in block B_j and $j > 2i$. Consider $f(x)$, its length is at least t_{2i}^k. We can now output a string that does not belong to L as follows: by cycling through all strings of length t_{2i} find a string x that does not belong to L. Output $f(x)$. This takes less than $2^{2t_{2i}^k}$ time, and it follows that $f(x)$ does not belong to L. Since $f(x)$ belongs to block B_j and $j > 2i$, it must be the case that $m = |f(x)| \geq t_{2i}^k$. Thus the time taken to output $f(x)$ (a string of length m) is at most 2^{2m}.

Thus for every string of length t_{2i} that is not in L, we can output a string that is not in L. By our assumption, there exist infinitely many strings of length t_{2i} that are not in L, and so there is procedure that outputs infinitely many string that are not in L.

In the actual proof, outputting strings that are not in L does not suffice. We have to output strings on which L differs from $\overline{\mathrm{SAT}}$, i.e., strings that are not in $\overline{\mathrm{SAT} \cup L}$. This presents additional complications. For this we define three additional sets: an NP-complete set SAT_J and two sets in NP L_J and L_O that are disjoint from SAT. We will show that if there is a reduction from L_O to $\mathrm{SAT}_J \cup L_J$, then one can use this reduction to output strings that are not in $\overline{\mathrm{SAT} \cup L}$. Now we present a formal proof.

Let $t_1 = 2$ and $t_{i+1} = t_i^{k^2}$. Consider the following sets.

$$\begin{aligned} E &= \{x \mid \exists i > 0 \text{ such that } x \in B_{2i}\} \\ O &= \{x \mid \exists i > 0 \text{ such that } x \in B_{2i+1}\} \\ J &= \{x \mid |x| = t_i \text{ and } i \text{ is even}\} \end{aligned}$$

Note that J is a subset of E.

If NP = co-NP, then SAT and $\overline{\mathrm{SAT}}$ are NP-complete and their union is Σ^*. The set Σ^* can not be complete for any class under many-one reductions. Let us assume that NP $\neq$ co-NP. Then, it must be the case that $\overline{\mathrm{SAT} \cup L}$ is infinite. Since $E \cup O = \Sigma^{\geq 2}$, at least one of $E \cap (\overline{\mathrm{SAT} \cup L})$ or $O \cap (\overline{\mathrm{SAT} \cup L})$ is infinite. From now on we will assume that $O \cap (\overline{\mathrm{SAT} \cup L})$ is infinite. If that were not the case we can interchange the roles of E and O, take $J = \{x \mid |x| = t_i \text{ and } i \text{ odd}\}$, and the proof structure remains similar.

Let $L_J = L \cap J$, $L_O = L \cap O$, and $\mathrm{SAT}_J = \mathrm{SAT} \cap J$.

Lemma 3.3. *The set* $\mathrm{SAT}_J \cup L_J$ *is not* NP*-complete.*

Observe that SAT_J is NP-complete. Clearly, L_J is disjoint from SAT_J. Thus by Observation 3.1, the theorem is implied by Lemma 3.3. The rest of the proof is dedicated to proving the above lemma.

Our proof proceeds by contradiction. Suppose $\mathrm{SAT}_J \cup L_J$ is NP-complete. Since L_O is in NP, there is a polynomial-time many-one reduction f from L_O to $\mathrm{SAT}_J \cup L_J$. Using this reduction f, we exhibit a 2^{2n}-time bounded refuter R that distinguishes $\overline{\mathrm{SAT}}$ from L. This contradicts Statement 1.

Let

$$T = O \cap (\overline{\mathrm{SAT} \cup L}) = O \cap \overline{\mathrm{SAT}} \cap \overline{L}.$$

Recall that T is infinite. Consider the following sets.

$$\begin{aligned} T_1 &= \{x \in T \mid f(x) \notin J\} \\ T_2 &= \{x \in T \mid f(x) \in J \text{ and } |f(x)| < |x|\} \\ T_3 &= \{x \in T \mid f(x) \in J \text{ and } |f(x)| \geq |x|\} \end{aligned}$$

Clearly $T = T_1 \cup T_2 \cup T_3$. We now show that each of T_1, T_2, and T_3 is finite. Since T is infinite, we obtain a contradiction.

Lemma 3.4. *T_1 is finite.*

Proof. Suppose not. Since $T \subseteq \overline{\mathrm{SAT} \cup L}$, T_1 is an infinite subset of $\overline{\mathrm{SAT} \cup L}$. Consider the following refuter R.

```
1. Input $1^n$.
2. For every $x \in \Sigma^n$ do
    (a) If $x \notin O \cap \overline{\mathrm{SAT}}$, then go to the next $x$. Else compute $f(x)$.
    (b) If $f(x) \notin J$, output $x$ and stop. Else go to the next $x$.
3. Output $\perp$.
```

The algorithm considers at most 2^n strings x. Since f is polynomial-time computable and SAT is in DTIME(2^n), R runs in time 2^{2n}. We now claim that R distinguishes $\overline{\mathrm{SAT}}$ from L. Consider an input length n.

Claim 3.5. *If $z \notin T_1 \cap \Sigma^n$, then $R(1^n)$ does not output z.*

Proof. If $z \notin T_1$, then either $z \notin T$ or $f(z) \in J$. Note that the above refuter outputs a string x only when $f(x) \notin J$. Thus if $f(z) \in J$, then it does not output z. Now consider the case $z \notin T$. If $z \notin O \cap \overline{\mathrm{SAT}}$, then the refuter does not output z. So assume $z \in O \cap \overline{\mathrm{SAT}}$. Since $z \notin T$, it follows that $z \in L$. Since $z \in O$ and $z \in L$, $z \in L_O$. If $z \in L_O$, then $f(z) \in \mathrm{SAT}_J \cup L_J$ and thus $f(z) \in J$. Thus the above refuter does not output z. □

Claim 3.6. *If $T_1 \cap \Sigma^n$ is not empty, then $R(1^n)$ outputs a string from $T_1 \cap \Sigma^n$.*

Proof. Let y be the lexicographically first string from $T_1 \cap \Sigma^n$. By the previous claim, $R(1^n)$ does not output any $z < y$. Thus the loop of the above algorithm considers y. Since $y \in T_1$, both the conditions $y \in O \cap \overline{\mathrm{SAT}}$ and $f(y) \notin J$ are satisfied. When this happens the refuter outputs y. □

Thus for every n, $R(1^n)$ either outputs $\perp$ or outputs a string from T_1. If T_1 is infinite, then for infinitely many such n, $R(1^n)$ outputs a string from T_1. Since T_1 is a subset of T and T is a subset of $\overline{\mathrm{SAT} \cup L} = L\Delta\overline{\mathrm{SAT}}$, it follows that the output of R belongs to $L\Delta\overline{\mathrm{SAT}}$. This contradicts Statement 1. Thus T_1 is a finite set, which proves Lemma 3.4. □

Lemma 3.7. *T_2 is finite.*

Proof. If T_2 is infinite, then T_2 is an infinite subset of $\overline{\mathrm{SAT} \cup L}$. Consider the following refuter R.

```
1. Input $1^n$.
2. For every $x \in \Sigma^n$ do
    (a) If $x$ does not belong to $O \cap \overline{\mathrm{SAT}}$, then go to the next $x$. Else, compute $f(x)$.
    (b) If $f(x) \notin J$ or $|f(x)| \geq |x|$, then go to the next $x$.
    (c) If $f(x) \in J$ and $|f(x)| < |x|$, then output $x$ if $f(x) \notin \mathrm{SAT}_J \cup L_J$. Else go to the next $x$.
```

3. Output $\perp$.

Checking whether $x \in O \cap \overline{\text{SAT}}$ takes time $O(2^n)$. Now we argue that checking the membership of $f(x)$ in $\text{SAT}_J \cup L_J$ takes $2^{|x|}$ time. We will check whether $f(x)$ is in $\text{SAT}_J \cup L_J$ only when $|f(x)| < |x|$ and $f(x) \in J$. By Step 2a, x is in O. Thus $t_i^{1/k} \leq |x| < t_i^k$ for some odd i. Since $f(x) \in J$, $|f(x)| = t_j$ for some even j. Since $|f(x)| < |x|$ and the interval $[t_i^{1/k}, t_i^k)$ contains exactly one t_l and this unique member is t_i, it follows that $j < i$.

Thus $t_i \geq t_j^{k^2}$ and so $t_i^{1/k} \geq t_j^k$. Since $|f(x)| = t_j$ and $|x| \geq t_i^{1/k}$, it follows that $|f(x)| \leq |x|^{1/k}$. Since L_J is decidable in time 2^{n^k} and SAT_J is decidable in time 2^n, we can decide the membership of $f(x)$ in $\text{SAT}_J \cup L_J$ in time $O(2^{|x|})$.

The algorithm checks whether $f(x)$ is in $\text{SAT}_J \cup L_J$ only when $|f(x)| < |x|$. Thus the total time taken by the above refuter is at most 2^{2n}.

Let n be the input length. As before, we make two claims.

Claim 3.8. *If $z \notin T_2 \cap \Sigma^n$, then $R(1^n)$ does not output z.*

Proof. If z does not belong to T_2, then either $z \notin T$ or $f(z) \notin J$ or $|f(x)| \geq |x|$. If $f(z) \notin J$ or $|f(z)| \geq |z|$, the refuter does not output z. Suppose $z \notin T$.

If $z \notin O \cap \overline{\text{SAT}}$, the refuter does not output z. If $z \in O \cap \overline{\text{SAT}}$, it follows that $z \in L$. Since $z \in O$, it follows that $z \in L_O$. Thus $f(z) \in \text{SAT}_J \cup L_J$. The refuter does not output any string z such that $f(z)$ belongs to $\text{SAT}_J \cup L_J$. Thus the refuter does not output z. $\square$

Claim 3.9. *If $T_2 \cap \Sigma^n \neq \emptyset$, then $R(1^n)$ outputs a string from $T_2 \cap \Sigma^n$.*

Proof. Let y be the lexicographically first string from $T_2 \cap \Sigma^n$. Let z be a string of length n that is smaller than y. By the previous claim, $R(1^n)$ does not output z. So the above refuter considers y during some iteration. Since $y \in T_2$, it must be the case that $y \in T$, $f(y) \in J$, and $|f(y)| < |y|$. If $y \in T_2$, then $y \notin L_O$. Since f is reduction from L_O to $\text{SAT}_J \cup L_J$, $f(y) \notin \text{SAT}_J \cup L_J$. Thus the refuter outputs z. $\square$

Thus for every n, $R(1^n)$ either outputs $\perp$ or outputs a string from $T_2 \cap \Sigma^n$. If T_2 is infinite then, for infinitely many n, $T_2 \cap \Sigma^n$ is not empty. Thus for infinitely many n, the refuter on input 1^n outputs a string from T_2. Since T_2 is a subset of $\overline{\text{SAT} \cup L}$, the output of $R(1^n)$ belongs to $L \Delta \overline{\text{SAT}}$. This is a contradiction. Thus T_2 is also finite, which proves Lemma 3.7. $\square$

We now claim that T_3 must also be finite.

Lemma 3.10. *T_3 is finite.*

Proof. Consider the following refuter R.

1. Input 1^n.
2. If $n = t_i$ for some even i, then proceed to step 3. Otherwise, output $\perp$.
3. For each y, $|y| \leq n^{1/k}$, test whether $|f(y)| = t_i$ and $y \in T$. If there is no such y, output $\perp$.
4. Let Y be the set of all y's that pass the test and let $X = \{f(y) \mid y \in Y\}$. Output one y such that $f(y)$ is the smallest member of X.

We first analyze the running time of R. There are at most $2 \cdot 2^{n^{1/k}}$ strings y that the refuter considers in step 3. Moreover, f is polynomial-time computable and T is decidable in time $O(2^{n^k})$. Therefore, the test in step 3 can be carried out in time $2^{n+n^{1/k}} < 2^{2n}$, which is a bound for the total run time of the refuter R.

Let y be a string from T_3. Since $y \in T_3$, there exists an odd number i such that the length of y lies between $t_i^{1/k}$ and t_i^k. By the definition of T_3, we have that $f(y) \in J$ and $|f(y)| \geq |y|$. From this it follows that there exists an even number $r > i$ such that $|f(y)| = t_r$. Since $r > i$ and $t_r = t_{r-1}^{k^2}$, it follows that $|f(y)| \geq |y|^k$.

Let n be a length at which T_3 is not empty. Let m be the smallest number such that $f(T_3 \cap \Sigma^n) \cap \Sigma^m \neq \emptyset$. By the previous discussion, it follows that $m \geq n^k$. Let z be the lexicographically smallest string from $f(T_3) \cap \Sigma^m$.

Claim 3.11. *If a string x of length m does not belong to $f(T_3)$, then $R(1^m)$ does not output x.*

Proof. If $x \notin J$ or there is no string y of length at most $m^{1/k}$ for which $f(y) = x$, then clearly $R(1^m)$ does not output x. Let us assume that $x \in J$ and there is a y for which $f(y) = x$. Observe that R outputs x only when $y \in T$. Since $f(y) = x \in J$ and $|f(y)| \geq |y|$, $y \in T$ implies that $y \in T_3$. However x is not in $f(T_3)$. Thus $y \notin T$ and so R does not output x. □

Claim 3.12. *If z is the lexicographically smallest string from $f(T_3) \cap \Sigma^m$, then $R(1^m)$ outputs z.*

Proof. By previous claim, $R(1^m)$ does not output any string smaller than z. Thus it considers z during some iteration. Since $z \in f(T_3)$, we have that $z \in J$. Let y be a string from T_3 such that $f(y) = z$. By our previous discussion, $|y| \leq m^{1/k}$. Since $y \in T_3$, $y \in T$. Thus the refuter outputs z. □

Thus every output of R is either $\perp$ or a string z from $f(T_3)$. Since $T_3 \subseteq T$, $T \subseteq \overline{L_O}$, and f is a many-one reduction from L_O to $\mathrm{SAT}_J \cup L_J$, it follows that $z \notin \mathrm{SAT}_J \cup L_J$ for each such z. Since $z \in J$, we have that $z \in \overline{\mathrm{SAT}\Delta L}$.

If T_3 is infinite, then $f(T_3)$ is also infinite. Thus for every m, $R(1^m)$ outputs $\perp$ or outputs a string from $\overline{\mathrm{SAT}\Delta L}$ and for infinitely many m, $R(1^m) \in \overline{\mathrm{SAT}\Delta L}$. This contradicts Statement 1 and so T_3 is a finite set, which proves Lemma 3.10. □

Thus it follows that T must be a finite set, which is a contradiction. Thus f can not be a many-one reduction from L_o to $\mathrm{SAT}_J \cup L_J$. Thus $\mathrm{SAT}_J \cup L_J$ is not many-one complete. This finishes the proof of Theorem 3.2. □

We will now show that if Statement 2 is true, then NP-complete sets are closed under disjoint unions.

Let A and B be two disjoint NP-complete sets whose union is not NP-complete. Consider $A \times \Sigma^*$ and $B \times \Sigma^*$. These sets are disjoint and are NP-complete. Also, their union is not NP-complete. Since $A \times \Sigma^*$ and $B \times \Sigma^*$ are paddable, they are isomorphic to SAT [BH77]. Thus if there exist two disjoint NP-complete sets A and B such that $A \cup B$ is not NP-complete, then there exist two disjoint NP-complete sets C and D that are isomorphic to SAT such that $C \cup D$ is not NP-complete.

Since C is isomorphic to SAT, there is a polynomial-time invertible bijection f from Σ^* to Σ^* that is a reduction from C to SAT. Now consider the sets SAT and $f(D)$. Since f is polynomial-time invertible, $f(D)$ belongs to NP. Moreover $f(D)$ is disjoint from SAT. Suppose there is a reduction

g from SAT to SAT $\cup f(D)$, then $f^{-1}g$ is a reduction from SAT to $C \cup D$. Thus if SAT $\cup f(D)$ is NP-complete, so is $C \cup D$. Thus we have the following observation.

Observation 3.13. *If there exist two disjoint* NP*-complete sets whose union is not* NP*-complete, then there is a set B in* NP *that is disjoint from* SAT *such that* SAT $\cup B$ *is not* NP*-complete.*

Theorem 3.14. *If Statement 2 is true, then unions of disjoint* NP*-complete sets are* NP*-complete.*

Proof. By the previous observation, it suffices to show that if L is any set in NP that is disjoint from SAT, then $L \cup$ SAT is NP-complete.

Consider the following set

$$B = \{x \mid \exists\, y, |y| \leq |x|, \text{ and } x \vee y \in L\},$$

where $x \vee y$ denotes the disjunction of the boolean formulas x and y. Clearly $B \in$ NP, and is disjoint from SAT. Thus by our Statement, there is a polynomial-time bounded refuter R such that for all but finitely many n, $R(1^n) \in \overline{\text{SAT}}\Delta B$. Since $B \subseteq \overline{\text{SAT}}$, $R(1^n) \in \overline{\text{SAT} \cup B}$.

Consider the following reduction from SAT to SAT $\cup L$. On input x, let y be the string having some length m such that $y = R(1^{|x|})$. Output $y \vee x$.

Since y does not belong to SAT, $x \in$ SAT if and only if $(y \vee x) \in$ SAT. It remains to show that if x is not in SAT, then $y \vee x$ is not in L: Suppose $x \notin$ SAT and $y \vee x \in L$. Then by the definition of B, y must belong to B. However y belongs to $\overline{\text{SAT} \cup B}$, which is a contradiction. □

3.1 Length-Increasing Reductions

As mentioned in the preliminaries our two statements are not converses of each other. Thus our sufficient and necessary conditions are not equivalent. Ideally, we would like to make them equivalent. We observe that if we strengthen the notion of NP-completeness to "completeness via length-increasing reductions," then we can make the necessary and sufficient conditions to be equivalent. Consider the following question: is the union of disjoint NP-complete sets complete via length-increasing reductions?

Theorem 3.15. *Unions of disjoint* NP*-complete sets are* NP*-complete under length-increasing reductions if and only if Statement 2 is true.*

Proof. As before, it is easy to see that there exist disjoint NP-complete sets whose union is NP-complete via length-increasing reductions if and only if for every set $B \in$ NP that is disjoint from SAT it holds that SAT $\cup B$ is complete via length-increasing reductions.

Let B be a set in NP that is disjoint from SAT. Let f be a length-increasing polynomial-time many-one reduction from SAT to SAT$\cup B$. Consider the following refuter R. On input 1^n generate an unsatisfiable formula ϕ of length $\geq n$. Output $f(\phi)$. Since generating unsatisfiable formulas is easy, the refuter runs in polynomial time. Since $\phi \notin$ SAT, $f(\phi) \notin$ SAT $\cup B$. Thus $f(\phi)$ is an unsatisfiable formula that does not belong to B, i.e., $f(\phi) \in B\Delta\overline{\text{SAT}}$. Since the length of ϕ is at least n and f is length-increasing, the length of $f(\phi)$ is at least n. Thus R almost everywhere distinguishes $\overline{\text{SAT}}$ from B.

The other direction follows from the proof of Theorem 3.14 as the reduction exhibited in that proof is length-increasing. □

Agrawal [Agr02] showed that if one-way permutations exist and E does not have $2^{\epsilon n}$ size circuits, then NP-complete sets are complete via length-increasing reductions. This yields the following corollary.

Corollary 3.16. *Assume that one-way permutations exist and there is a language in* E *that requires* $2^{\epsilon n}$*-size circuits for some* $\epsilon > 0$. *Then unions of disjoint* NP*-completes are* NP*-complete if and only if Statement 2 is true.*

4 Discussion

Suppose NP $\neq$ co-NP and let L be a language in NP that is disjoint from SAT. Since L is in NP L can be decided in time 2^{n^k}. Since L does not equal $\overline{\text{SAT}}$, there exists a refuter that distinguishes L from $\overline{\text{SAT}}$. What is the complexity of such a refuter? It is easy to see that there is a refuter that distinguishes L from $\overline{\text{SAT}}$ and this refuter runs in time $2^{n^{k+1}}$. Statement 1 implies that there is no refuter whose running time is drastically better whereas Statement 2 implies that there is a refuter that runs in polynomial time.

Our results indicate that to settle the question of whether unions of disjoint NP-complete sets remain NP-complete, one must understand the complexity of refuters. We have provided necessary and sufficient conditions for the answer to this question to be true. Clearly there is a gap between the necessary and sufficient conditions. We can bridge this gap under certain believable hypotheses. It would be interesting to bridge it unconditionally.

5 Acknowledgments

We thank the anonymous referees for numerous helpful suggestions and comments. We thank Eric Allender for pointing out a gap in the proof of one of the claims.

References

[Agr02] M. Agrawal. Pseudo-random generators and structure of complete degrees. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 139–145. IEEE Computer Society, 2002.

[BH77] L. Berman and H. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[ESY84] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

[FLM10] L. Fortnow, J. H. Lutz, and E. Mayordomo. Inseparability and strong hypotheses for disjoint NP pairs. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, volume 5 of *Leibniz International Proceedings in Informatics*, pages 395–404. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[Gol06] O. Goldreich. On promise problems: A survey. In *Theoretical Computer Science - Essays in Memory of Shimon Even*, pages 254–290. Springer, 2006.

[GPSS06] C. Glaßer, A. Pavan, A. Selman, and S. Sengupta. Properties of NP-complete sets. *SIAM Journal on Computing*, 36(2):516–542, 2006.

[GPSZ08] C. Glaßer, A. Pavan, A. Selman, and L. Zhang. Splitting NP-complete sets. *SIAM Journal on Computing*, 37(5):1517–1535, 2008.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–355, April 1988.

[GSS05] C. Glaßer, A. Selman, and S. Sengupta. Reductions between disjoint NP-pairs. *Information and Computation*, 200(2):247–267, 2005.

[GSTW08] C. Glaßer, A. Selman, S. Travers, and K. W. Wagner. The complexity of unions of disjoint sets. *Journal of Computer and System Sciences*, 74(7):1173–1187, 2008.

[GSZ07] C. Glaßer, A. Selman, and L. Zhang. Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science*, 370(1):60–73, 2007.

[IW01] R. Impagliazzo and A. Wigderson. Randomness vs. time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63:672–688, 2001.

[JS05] J. Jackson and R. Servedio. On learning random DNF formulas under the uniform distribution. In *Proceedings of the 9th International Workshop on Randomness and Computation*, pages 342–353. Springer, 2005.

[Kab01] V. Kabanets. Easiness assumptions and hardness tests: trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001.

[Pud01] P. Pudlak. On reducibility and symmetry of disjoint NP-pairs. Technical Report TR01-044, Electronic Colloquium on Computational Complexity, 2001.

[Raz94] A. Razborov. On provably disjoint NP pairs. Technical Report TR94-006, Electronic Colloquium on Computational Complexity, 1994.

[San09] R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. *SIAM Journal on Computing*, 39(3):1038–1061, 2009.

[Sel88] A. Selman. Natural self-reducible sets. *SIAM Journal on Computing*, 17(5):989–996, 1988.